The Expert's Guide to SQL Expressions, Options and Commands

Copyright © 2007 by Ken Hamady
525K East Market St. PMB 299
Leesburg, VA 20176
(540) 338-0194
www.kenhamady.com (web site)
ken@kenhamady.com (email address)

Fair use of electronic books
Whenever someone publishes an electronic book, they take the risk that users will make extra copies of the material and share copies with co-workers and friends. These same readers wouldn't buy a book at the bookstore and photocopy the entire book for a friend, but copying a file with one click doesn't seem to be the same thing … but of course it is. It certainly wouldn't be fair for a company to buy one copy of this book, and give copies of the PDF file to their entire technical team, or place it on a server for everyone to use simultaneously.

Of course you can let someone borrow a copy of this book, just like you can with regular printed book. But if you give someone a copy of this book and you keep using the original copy, then you really are making a second copy of the material. Here is a simple test to see if you are using the material fairly. Print out one copy of the material and limit yourself and your colleagues to using that one hard copy. Pass that single hard copy around as much as you like. If sharing one hard copy is a burden, then you probably should be purchasing more than one copy of the material. If you want a price for multiple copies or a site license for the material, please let me know.

Notes on the Sample Reports
The sample reports are used as examples and as a starting point for several exercises. Many of the reports were created using version 8.5 so that as many users as possible can open them. Reports that demonstrate features available only in later versions (Commands, Dynamic parameters) are created in the earliest version that supports that feature.

Most sample reports connect to the Xtreme sample database so they use the SQL syntax for Microsoft Access. I have included three additional sample reports that illustrate SQL Expressions for three other common SQL variations: SQL Server, Oracle and MySQL. See Appendix A for key syntax differences.

All sample reports should open in preview because they have saved data. If you refresh the Xtreme based reports, Crystal will ask you to select your local Xtreme DSN. If you do not have the Xtreme DSN search my blog for "Xtreme" for instructions. You will not be able to refresh the other reports. You can open and view the SQL Expressions, but when you do Crystal will ask you to select a local DSN. Select any DSN for any database to view the SQL Expressions. But if you try to save or change these expressions the new DSN will generate an error because the fields do not exist locally. It is best to use these as examples for creating expressions in your own reports.

Acknowledgements

I would like to thank two people for their assistance with this material. First would be Joel Seguin, Sr. Analyst at Health Net of California. He took the time to review the first draft and suggested the subreport as a method for adding a multi-value parameter to a command; a very clever solution.

And then there is Linda Bass, who uses the handle LBass in the Crystal Reports forums on Tek-Tips.com. It was LBass that first introduced me to subqueries within SQL expressions. She planted the seeds that eventually grew into this guide. I was therefore very pleased when Linda agreed to review the material. She went further and tested all the exercises, found many typos and suggested several significant additions. She provided my first example of a filtered Left Outer join using the FROM clause. She and Joel then worked together in Tek-Tips to uncover the most efficient way to assign an alias to a subquery within a SQL expression.

And, if you are serious about Crystal Reports then you should visit Tek-Tips.com. It is the most active Crystal Reports forum on the web. And there are several experts, like Linda, that work hard to keep it humming.

Table of Contents

What is SQL

SQL is the Structured Query Language, a relatively universal language that can be used to request data from most databases. In most reports, Crystal will automatically create and send a SQL request for the user, based on the needs of the report. The database sends the requested data back to Crystal in a results table. This table is the starting point for every report. This guide is designed to help you understand how Crystal generates those SQL requests. I will also show you how you can improve those requests or even replace them altogether.

To see what a SQL (pronounced like "sequel") request looks like, open a report that was created using SQL based data and use the menu options **Database > Show SQL Query**. What you see is called the SQL statement. This is the request that is sent to the database. If this menu item isn't available then the report is not generating a SQL request.

The basic SQL Statement has up to five sections. The first two statements (SELECT and FROM) will always appear. The other statements appear when they are needed.

1) The "SELECT" clause lists the fields that will be retrieved from the data tables.
2) The "FROM" clause lists the tables that will be used, and how they are related.
3) The "WHERE" clause lists criteria for selecting records.
4) The "ORDER BY" puts the records in order, based on the grouping and sorting in the report.
5) The "GROUP BY" causes the database to return only subtotals for each "GROUP BY" value.

Reporting on Views and Stored Procedures

Before we talk about how Crystal generates the SQL, I should mention the one option that eliminates any need to discuss SQL within Crystal Reports. That is to store all of the SQL queries in the database. The two options for doing this are usually called views and stored procedures, although MS Access calls them queries. Users who are very proficient in SQL, and who have the rights to create database objects, will often write all of their queries in the database and store them as views or stored procedures. Crystal Reports becomes primarily a formatting and printing tool for the final presentation. This material does not teach you how to create views or stored procedures in a database. See Appendix C for a few notes on using views and stored procedures with Crystal Reports.

How Crystal Reports options translate into SQL clauses

SELECT clause (fields used)

Every field that you use in a report will show up in the SELECT clause. This includes fields visible on the output as well as the invisible fields used for things like selecting records, sorting and writing formulas.

FROM clause (tables and links used)

Tables

Every table you add to a report will show up in the FROM clause, as long as the table is used. There are three ways that a table can be used:

1) Use a field from that table on the report or in a feature of the report
2) Link that table between two other tables that are both used in the report
3) Tell Crystal to "Enforce the Join" for that table (discussed below)

If a table is added to the report but not used in any of these three ways it will not show up in the FROM clause. I call these "sleeping tables" and recommend that you avoid them. Often reports will work fine for a while, but then will do strange things when the first field is used from a sleeping table. That first field activates the table and its joins. Only then do you know if that table is compatible with your report or if it should be removed.

Joins (Links)

With very few exceptions, every table added to a report has to be joined to the other tables. These joins, and their related options will also show up in the FROM clause of the SQL statement. This includes the choice of INNER or OUTER joins, and also the comparison type that is used for each join. There is an entire section below on creating joins and setting their options.

Record Selection - the WHERE clause

In Crystal Reports you set the criteria for the records in either the Selection Formula, or in the Select Expert (which then writes the selection formula). Crystal will attempt to convert some or all of that formula into the WHERE clause of the SQL statement. But not all Crystal criteria can be converted by Crystal into SQL. When this happens the database sends an initial results table with records that meet the converted criteria. Usually this has too many rows. Crystal then has to apply the record selection formula to the records returned. This ensures that the records in the final results table meet all of the criteria for the report.

Grouping and Sorting - the ORDER BY clause

Crystal will attempt to add the groups and sorts assigned to the report to the ORDER BY clause so that the records are sorted by the database. This means that Crystal doesn't have to do the sort locally. However if the Groups and Sorts can't be converted to SQL (for example, if you group or sort on a formula) then the ORDER BY will be missing and the sorting will happen in Crystal.

Performing Grouping on the Server - the GROUP BY clause

If you design your report correctly you can get the database to do the grouping and subtotals for you. You will know this has happened if you see a GROUP BY clause at the end of the SQL statement. This is discussed in more detail below.

Improving the Automatically Generated SQL

Most reports run fine using the automatically generated SQL, but there are ways to make reports run faster by knowing how to improve the SQL request that Crystal creates. And there are some report requirements that can only be met by going beyond the SQL that Crystal generates and using more advanced SQL queries. Some users simply prefer to write their own queries.

Recent versions of Crystal Reports (v9 or later) only allow you to view or copy the auto-generated SQL statement. No changes are allowed in the "Show SQL" window. But these same versions allow you to create your own statement, from scratch, by using a SQL "command" as the starting point of the report. Commands are covered in a later section of this guide.

Older versions (8.5 and prior) allow limited changes to the auto-generated SQL. This is also covered in a later section.

Perform Grouping on Server

This option is in Database menu and also under File >Report Options. It is designed to add the GROUP BY clause to your SQL Statement. This causes the database to summarize the data for you. The database will only return one record for each group rather then sending the details. However simply setting this option is rarely enough to have the SQL add the Group By. This option only works if the report meets several specific criteria.

Things you must do:
Group on a database field or a SQL expression (not a formula)
Hide or suppress the details
Make all visible fields either a group field or a summary field
(For other detail fields create a Maximum summary field to display)
Limit the summaries used to those supported by the Database
(Sum, Count, Minimum and Maximum should be safe)

Things you must not do:
Add Specified Order Grouping.
Add Running Totals on detail fields.
Add Summaries of formulas fields.

If you report meets these criteria Crystal should be able to add a GROUP BY to your SQL expression and get the server to group the records calculate all of the summary fields. Always check to make sure the GROUP BY appears.

Select Distinct Records

You can ask Crystal to change the first clause from "SELECT" to "SELECT DISTINCT". You activate this by using **Database > Select Distinct Records** or under **File > Report Options**. It Activating this feature will eliminate all duplicate records from your results table - but only if ALL fields selected for the report are the same in both records. If two records are similar, but one field is different, they will not be considered duplicates. Both will show up in your results.

Exercise 1 - SQL Basics

Open the file Master.rpt.
Preview the report and select your local connection for the Xtreme Sample Database.
Save the report (as Master).
Save it again as Lesson 1.

Use the menu commands Database > Show SQL Query and note the ORDER BY.
Close the window and add a Record Sort using Order Date.
Show the SQL again and note the change in the ORDER BY.
Save the report.

Save the report again as Lesson 1A.
Hide the Details section and the Group Header.
Refresh the report and check record count (lower right corner).
Go into File > Report Options and check "Perform Grouping on Server".
(This is also available in the Database menu.)
Click OK and note the change in the record count.
Show the SQL and note the GROUP BY has been added.

Double click on any subtotal to drill-down on that group.
Note the record count for that group.
Show the SQL while in Drill down and note the change in the WHERE clause.
Save the report.

Open Lesson 1B.
Preview and set the location to your local Xtreme connection.
Note the Record Count.
Go into the Database Menu and check "Select Distinct Records".
Refresh and show SQL.
Note the second word is now DISTINCT and the Record count drops slightly.

Go into the Database menu and UNcheck "Select Distinct Records".
Go to Design mode and delete all four objects in the last column, including the totals.
Preview and note the number of records goes back to the original number.
Go into the Database Menu and again check "Select Distinct Records".
Note the drastic drop in the record count even though both tables are being used.
Insert the Amount field back onto the report.
Note the increase in records again.

Understanding Joins

What joins (links) do

Most databases store data that is frequently repeated in separate tables. This minimizes the amount of redundant data that is stored. Joins are used to combine these separate tables back into one composite table.

Creating a Join

You create joins on the "links" tab of the Database Expert (which was called the "Visual Linking Expert" before v9). To create a join you drag the linking field from one table and drop it on the corresponding field in the other table. The table you drag from is named the LEFT or FROM table of the join and the other table is named the RIGHT or TO table. This distinction is not important when the join is first created, but will affect the join options discussed below. To check the direction of an existing join you right-click on the line between those two tables and select "Link Options". As a rule of thumb it is best to have all of the joins in a report going in one direction, left to right, in the links window.

Some joins require that you link multiple fields between the same two tables. The direction that you drag the first field determines the direction of the entire join. Subsequently, you could right-click on any of the lines between the two tables to "reverse" the entire join.

Auto ("Smart") Linking

Crystal has a feature that adds links automatically whenever you add tables. Unfortunately this feature usually does NOT get the correct links, so you must double check the automatic links and change them if they are not correct. If you don't know what the correct links are then you may have to refer to the documentation provided by the person/vendor who created the database tables, or you may have to identify the correct links by trial and error. This feature can be turned off in the "Database Options" of most versions of Crystal.

Inner and Outer Joins

When you first make a link it is always set to be an "Inner" join by default (called an "Equal" join before v9). An Inner Join tells the database that the records in the Left table must find a matching record in the Right table before the Left record can be included on the report. The reverse is also true. Any record in the Right table must have a match in the Left table or it will not be included in the query results. I would estimate that 90% of the joins in my reports are Inner Joins.

But there are some situations where the report should include unmatched records. In these cases you are allowed to change the Inner Join to one of the "Outer Join" options. There are three types:

Left Outer Join - This is the most common of the three. It tells the database to show all records from the LEFT (or FROM) table and only show records from the RIGHT table if they match the linking field in the LEFT table. The RIGHT table becomes optional. When LEFT records have no match in the RIGHT table the requested columns from the RIGHT table are left as NULL values.

Right Outer Join - Tells the databases to show all records from the RIGHT or TO table and only show records from the LEFT or FROM table if they match the linking field in the RIGHT table. The LEFT table becomes optional. When RIGHT records have no match in the LEFT table, the requested columns from the LEFT table are left as NULL values.

Full Outer Join - Show all matching records and include all unmatched records on both sides. Both tables become Optional. All columns may have NULL values if there is no corresponding data.

"Breaking" an Outer Join

There are two additional requirements that you must keep in mind when using Outer Joins. Otherwise you will 'break' the Outer Join and it will behave like an Inner Join.

1) Any join that comes FROM an optional table must also be an Outer Join. Say you use a Left Outer Join to link from Table A to Table B. Then you use an Inner Join to link from table B to Table C. The Outer Join will behave like an Inner Join and Table B will no longer be optional.

2) If you add record selection criteria to the report using any field from an optional table, that table will no longer be optional. The link will now behave like an Inner Join. So in a Left Outer Join you could filter on the Left table (the FROM) but not on the Right table (the TO).

If multiple fields are used in a join, changing the join type for any field in that join will change the join type for the other fields in that join. There can only be one join type (Inner, Left Outer, etc) between a single pair of tables.

Enforcing Joins (v10 and later)

Above I mentioned "sleeping tables" which are tables that are in the Crystal Report but do not show up in the SQL statement because they are not used at all by the report. The simplest way to ensure a table is used is to put a field from that table onto the report. It doesn't even have to be visible. Another option is to tell Crystal to enforce the join for that table. In any join you can enforce the FROM table, the TO table or both. This forces the SQL to include the join for that table even when there are no fields being used from that table. If you don't use sleeping tables you don't have to worry about enforcing joins, so I haven't had to use this feature at all.

If multiple fields are used in a join, changing the enforcement setting for any field in that join will change the enforcement type for the other fields in that join. There can only be one enforcement type between two tables.

Join comparisons

Typically when you link two tables you are looking for records that match, meaning they have the same value for the linking field. So the most common comparison is Equal. The value in Table A equals the value in Table B. However there are a few situations where a different comparison is useful. For example, when you have a table that keeps a history of price changes for parts, and each record has an effective date and an expiration date. You could link the date of the transaction to both the effective date and the expiration date fields in the price table. You could set the link to the effective date to be >= while the link to the expiration would be <= which gives you the price in effect at the time of the transaction.

Unlike the previous two join options, the comparison option CAN be set differently for each field in a join.

Table Inflation

Two tables being linked will almost always have a "One-to-Many" relationship. This means that the linking field in one table will find several matching values in the other table (i.e. one Customer will find several Orders). When these two tables are linked, the record taken from the "one" table will link to ALL of the matching records in the "many" table. So when the results table is built for this report, the records from the 'one' table have to be repeated for each matching record found in the many table. So if a Customer has 10 orders, his Customer fields (like Customer name) will be included in the results 10 times, once for each order. Even numeric fields will repeat if they come from the 'one' end of a one-to-many relationship. I call this behavior "table inflation" because any sum that you do of these fields will be inflated by the repeated values. The records from the "many" table in the join do not repeat so totals using those columns are not inflated.

Note - to 'deflate' totals of inflated data you group the duplicates together and use running totals that evaluate 'once per group'. My Expert's Guide to Totals has an example of this technique.

Table Inflation can be a more serious problem when the 'one' table is linked to two different 'many' tables. Both of the "many" tables ends up inflating the 'one' table which means they inflate each other. Say you have Credits in one table and Debits in another. A customer record that links to three credits and five debits would be inflated into 15 records. Each Debit value would show up three times and each Credit value would show up five times. So in a situation like this simple linking won't work. See the section on UNIONS below for how to create a report with data like this.

Exercise 2 - Linking tables with a One-To-Many relationship

Open Lesson 2.
Preview and select your local connection of the Xtreme Sample Database.
Note the number of Employees, and the total salary.
Use the menu options **Database > Database Expert.**
(Use Database > Add Database in v8.x)
Select the table "Orders" and move it to the right.
Click "OK" and Crystal will link them for you.
(Or you can link them by Employee ID).
Click "OK" again and let Crystal Refresh the report.
Preview and note the totals haven't changed, yet.
(The Orders table is a sleeping table).
Switch to design mode.
Show the SQL and note that there is no mention of the Orders table.
Save this Report as Lesson 2A

Enforcing the Join (v10 and later)

Use the menu options **Database > Database Expert** and click on the "Links" tab.
Highlight the Join and click "Link Options".
Select "Enforce From" and click OK three times to refresh the data.
There is still no change in the report.
Go back to "Link Options" again.
Select "Enforce To" and click OK three times to refresh the data.
Note the huge increase in the number of records.
Show the SQL and note that the Orders table is mentioned in the SQL.

Go back to "Link Options" again.
Select "Not Enforced" and click OK three times to refresh the data.
The record count goes back to 15.
Show the SQL and note that the Orders table is now gone.

Switch to Design mode and open the Field Explorer.
Open the Orders table and place Order Amount on the details band.
Preview and note the increase in records.
Show the SQL and note the Orders table in the FROM clause .
(Using one field from a table is the same as enforcing the join for that table).
Save this Report as Lesson 2B

Change the join to an Outer Join

Jump to the last page of the report and look at the totals.
Note that the number of Employees listed on the report has dropped from 15 to 6.
(This is the result of the Inner join since some employees do not process orders).
Use the menu options Database > Database Expert and click on the "Links" tab.
Highlight the Join and click "Link Options".
Select "Left Outer Join" and click OK three times to refresh the data.
Go to the last page to view the totals.
Note that there are now 15 employees.
Save this Report as Lesson 2C

Seeing "Table Inflation

Note that the Salary column has duplicate information and an inflated total.
(This is the result of Table Inflation.).
To get an accurate total of an inflated column you can use a running total.
First, insert a group on the Employee ID.
Next, right-click on any detail value in the Salary column.
Select Insert > Running Total.
Change the middle setting (Evaluate) to be "On Change Of Group".
It should default to Group 1.
Click OK and move this object to the report footer.
It should show the original total of $668,000.
Save this Report as Lesson 2D
See "The Expert's Guide to Totals" for more information on running totals.

SQL Expressions

What are SQL Expressions

SQL Expressions are much like Crystal formulas, except that they use SQL syntax and SQL functions instead of Crystal syntax and functions. Often a formula can be replaced with a SQL expression that does the exact same thing. And SQL Expressions become part of the SELECT statement so they are processed by the database, not by Crystal Reports. In many situations they can speed up your reports by allowing the database to do more of the work and reducing the number of records returned to Crystal. They also allow you to do a few tricks that you can't do with formulas.

When to use SQL Expressions

The speed of most reports depends on how long it takes Crystal to get the needed records back from the database. And this depends, in turn, on how well Crystal can convert the record selection formula into the WHERE clause. As mentioned before, it is possible that some or all of the selection formula won't be converted into SQL. This means that the database will have to send back more records than the report requires. Crystal then eliminates these extra records by double checking the initial results against the record selection formula. The time spent to process these extra records (both by the database and by Crystal) can add significantly to the processing.

One way to make this more efficient is to change the selection formula so that all of it can be converted into SQL. This is especially important if your record selection formula has any Crystal functions in it, or contains formula fields that use functions. Often it is Crystal functions that prevent criteria from converting to SQL. The answer is to use SQL Expressions instead.

Reports that can't use SQL Expressions

Some reports don't allow SQL expressions and so the option will not appear in the Field Explorer. The following are the primary reasons why a report would have SQL expressions unavailable:

The report uses a non-SQL Data source (such as Xbase or Paradox)
The report is based on a Stored Procedure
The report is based on a Crystal command object
The report uses a command (unless the SQL Expression is added first)
The report connects to multiple data sources (unless the SQL Expression is added first)

Limitations of SQL Expressions

SQL Expressions have to be written in SQL Syntax, which can vary from database to database. So unlike formulas, a SQL Expression will often have to be rewritten if you change databases, say from SQL Server to Oracle.

One other serious limitation of SQL Expressions is that they can't incorporate parameters the way formulas and commands can. I am not sure why parameters are not available to SQL Expressions. SQL Expressions are calculated after the parameters are evaluated and parameters can go into SQL Commands. (We will have to suggest that to Crystal/BO/SAP).

How to Add and Use SQL Expressions

First open the Field Explorer, right-click on the category called "SQL Expression Fields" and select "New". (If the category doesn't appear, see the section above on reports that can't use SQL Expressions.) You will be asked to give the expression a name. When you click "OK" you will see an editor window, similar to the one used to write formulas. The main difference is that you have to use syntax, functions and operators that are supported by the database and ODBC driver used in this report. Because SQL expressions are database specific it is possible that a SQL Expression that works fine in one environment may not work in a different environment. Even going through ODBC can change the syntax. A statement that works in the database may not work the same when going through an ODBC connection.

Once a SQL expression is added to a report, you can use it like any other field. You can even use it within a formula. However you can't refer to one SQL expression field within another SQL expression.

Also, note that you get very little assistance from Crystal Reports when writing and testing your SQL expressions. Error messages that come back from the database are not always very helpful.

Using "Unlisted" Functions in SQL Expressions

When you first look at the SQL Expressions editor you might be discouraged by the short list of functions available. But unlike the formula editor, this list is not an exhaustive list of the available functions. You can use any function or operator that is accepted by the ODBC/SQL driver and the target database. For example, the following unlisted functions are used in my examples for the specified environments, and may well work in other environments:

Summary Functions: Max, Sum, etc work in all environments
Operators: Is Null, In and Between work in all environments

Here are other unlisted functions that I have tested in the specified environments.

- Cast - Oracle, SQL Server
- Val - Access
- CStr - Access
- CDate - Access
- DateAdd - Access, SQL Server
- DateDiff - SQL Server
- DatePart - Access
- Interval - MySQL, Oracle,
- Replace - SQL Server
- IF - MySQL
- IIF - Access
- Case - Oracle, SQL Server

For more syntax examples please refer to the four sample reports that are named:
"Lesson 3 My SQL", "Lesson 3 SQL Server", "Lesson 3 MS Access" and "Lesson 3 Oracle".

Writing Subqueries within SQL Expressions

It is even possible to do a separate SQL query (Select X from Y where Z) within a SQL Expression. As mentioned before, all SQL Expressions becomes part of the SELECT of the main query. So a query within a SQL Expression will become a subquery within the SELECT of the main query. And, if you incorporate the correct fields from the main query into the WHERE clause of the subquery you can correlate the subquery to the records of the main query. There are even times when this performs better than creating the entire query in a command object (see 'the downside of commands' in the section on command objects).

The main constraint within SQL Expressions is that each SQL Expression can only return a single value. So a subquery within a SQL Expression has to be designed to return only one* value. This usually means returning a summary value, like a sum or a maximum. Returning a summary using a subquery can help your deal with a one-to-many relationship that would otherwise generate table inflation in your report. It can also make summary values available for sorting or grouping while those same values, if calculated in CR, would not be available for sorting or grouping.

(*Note - there is a way to pack a combination of several values into the 'one' value returned by the expression. See the section below entitled 'returning multiple values in a SQL expression".)

Take a typical example. You have a table that stores orders for customers, and you want to group the records based on each customer's last order date. You could use Crystal to calculate the last order, but you wouldn't be able to group the records based on that value. However you could create a SQL Expression that calculated the date of the last order for each Customer. You could then group on that value.

There are two tricks to getting a SQL expression to be included as a subquery. The first is pretty simple. You have to put the entire SQL expression inside a pair of parentheses. These are transferred into the SELECT of the main query so that the expression is recognized as a subquery.

The second one has to do with creating a WHERE clause in the SQL Expression to 'correlate' the subquery to the records of the main query. Start with the following query that returns the last order date for one specific customer:

```
(Select Max(Orders.Date)
from Orders
where Orders.CustID = 14)
```

To get this query to "correlate" to your main query you have to replace the literal "14" in the WHERE clause with the Customer ID in each record of the main query. To do this you use a field from the main query that has the correlating value and put this on the right of the "=" sign. But here is the key: the field drawn from the main query has to come from a different table (or at least from a different table alias) then the corresponding field in the WHERE clause. Since the subquery above is drawing from the Orders table I can use the CustID from another table in the main query (like the Customer table) as the correlating field. So it would look like this:

```
(Select Max(Orders.Date)
from Orders
where Orders.CustID = Customer.CustID)
```

So the easiest method to use, the method I found to work across all versions of Crystal, and across all four databases that I tested, is to draw the correlating field from another table used in the main query. And, since the linking fields of the subquery are often also the linking fields between tables, it is usually not difficult to find another field in the main query that can be used for the WHERE of the subquery.

The reason this baffles many users is that now the subquery within SQL Expressions won't typically be valid as a standalone query or command. But it will work fine as a subquery because both of the tables are declared in the parent query.

Using aliases in the SQL expression

In some cases, though, there is no other table to provide the field for the WHERE clause. In these cases you can change the alias in the FROM of the subquery. This causes the SQL to treat the two table aliases as separate tables. In the example below I changed the alias for the Orders table to OrdAlias, and used the original table name for the correlating field in the WHERE clause. Note that you should use the field by itself within the summary function. Do not use either the original table name or the alias until you are in the FROM.

```
(Select Max(Date)
from Orders OrdAlias
where OrdAlias.CustID = Orders.CustID)
```

Using aliases in the main report

If you need to use the table name inside the summary function, maybe because there are multiple fields with the same name, then another alternative is to change the alias of the table in the MAIN report, and use the original table name in the FROM of the subquery. In the example below I changed the alias for the Orders table in the main report to OrdersMain, and used the original table name in the FROM of the subquery:

```
(Select Max(Orders.Date)
from Orders
where Orders.CustID = OrdersMain.CustID)
```

NOTE - I noticed one other special feature of CR 2008 (v12). Most prior versions will not allow you to query a table in a SQL expression if that table is not also used in the main report query. But in CR 2008 I was able to write a SQL expression that queried a table not otherwise used in the main report's SQL. I am not sure if this works in all environments but it was clear that reports that would run in CR 2008 would fail in XI because of this feature.

Returning multiple values in a SQL expression

As we said at the very beginning of this section, a SQL Expression can only return one value to the report. But, there is a way to return one value that actually contains a whole row of values. The idea comes from a technique I use in Crystal Reports called the Wormhole, which allows you to carry several values from the bottom of the report and show them at the top. The trick is to concatenate the values together before you transfer them and then split them apart afterwards. In a SQL expression you would write the query so that the SELECT asks for all of the values that you need, and then you concatenate them together separated by dashes or slashes. You then give an alias name to the concatenated string so that it is the 'one' value returned to the report. Last, you write formulas in the report to extract the individual values as separate fields.

In one special case I have included the concatenation within the SQL summary operation. One of my customers wanted to group students based on the category of their first contact with the school. So I wrote a SQL Expression that didn't just return the Minimum date for that person, but also brought along the text of that category. To do this I wrote an expression that converted the date and category into a string that looked like this:

20071231-category

The date is an 8 characters string in the format YYYYMMDD while the category starts in position 10. I then asked SQL to find the minimum value of this expression for each student. Because the date is the front of the expression, the minimum value will always be the first date. But because the category is concatenated, it will come along for the ride. Once that value is back in the main report I can split the string into two pieces and have both pieces of information available to use. And because SQL Expressions happen so early I can use these values for selecting, sorting, grouping and totaling.

Exercise 3 - Using SQL Expressions

3A) Using SQL Expressions instead of formulas

Open the master Report and save as Lesson 3A.
Add a new formula called "Left Initial" using the following formula:

```
Left ( {Customer.Customer Name} , 1 )
```

Add a new formula called "SubString Initial" using the following formula:

```
{Customer.Customer Name} [1]
```

Open the field explorer and right click on "SQL Expressions"
Give the new expression the name "SQL Initial"
From the function list open the category "String" and double click "Left".
Now double click the field Customer Name and add the number 1 so that it reads:

```
{fn LEFT(Customer.`Customer Name`,1 )}
```

Save and close the SQL Expression editor.

Open the Select Expert and delete any existing rules.
Add a new rule that says:
Left Initial / is equal to / B.
Preview the report and notice that only Customers that start with "B" are included.
Use the Menu Options **Database > Show SQL Query** to open the SQL window.
Note that the "Where" clause is missing, so all filtering is happening locally

Open the Select Expert and delete the "Left Initial" rule. Add a new rule:
Substring Initial / is equal to / B.
Refresh and notice that, again, only Customers that start with "B" are included.
Use the Menu Options **Database > Show SQL Query** to open the SQL window.
Note that the "Where" clause now includes the formula for Initial.
But since the comparison is ">=" most of the filtering is still happening locally.

Open the Select Expert and delete the rule for Substring Initial.
Add a new rule that uses the SQL Expression "SQL Initial":
SQL Initial / is equal to / B.

Note that SQL expressions are listed with **%** in front of the name instead of the **@.**
Refresh and notice that, again, only Customers that start with "B" are included.
Use the Menu Options **Database > Show SQL Query** to open the SQL window.
Note that WHERE includes the SQL Expression and the "=" comparison.

3B) Using Unlisted Functions

The following exercise is meant to be done in the Crystal Reports sample data based, so it uses MS Access Syntax. **For more syntax examples** please refer to the four sample reports that are named: "Lesson 3 My SQL", "Lesson 3 SQL Server", "Lesson 3 MS Access" and "Lesson 3 Oracle". Each contains 16 expressions showing the syntax for common calculations and one subquery example.

These reports are saved with data so they should open in preview mode. But before you can edit or view any SQL expression, Crystal will require that you connect to a DSN - any DSN for any database. Any change you make to one of these expressions, even if valid, is probably going to generate an error. The only exception will be the one for MS Access which should run fine against your local copy of the Xtreme sample database. For testing other syntax variations you should create equivalent expressions in your reports. Use the sample expressions for syntax examples.

Open Lesson 3B
Preview and select your local connection of the Xtreme Sample Database.
Add a new SQL Expression called IIF.
Type the expression below.

```
IIF (`Orders`.`Order Amount` > 1000 , 'Big', 'Small')
```

NOTE - The field can be selected from the field list.
Make sure you use "back ticks" for field names and single quotes for literals.
You will find back ticks next to the number 1, on the same key as the tilde (~) symbol.

Save this SQL Expression and put it on the details band.
Preview the report and note that all records under $1000 are labeled as Small.

Add a new SQL Expression called DateAdd.
Type the following expression:

```
DateAdd('m' , 3 , `Orders`.`Order Date`)
```

Save and put this on the details band.
Note that is shows a date 3 months after the order date.
Save the report.

3C) Writing Subqueries within SQL Expressions

Open Lesson 3B and save it as Lesson 3C.
Add a new SQL Expression called SubQuery1 using the syntax below (exactly):

```
(Select Max(`Order Date`)
from Orders
where `Customer ID` = Customer.`Customer ID`)
```

If you type the field names and tables make sure to use “back ticks” and not quotes.
Place this field on the Group Footer, just under the Order Date column.
Note that the last order date for this customer is printed.
Save the report.

3D) Subqueries that return several values

Open Lesson 3B and save it as Lesson 3D.
Add a new SQL Expression called SubQuery2 using the syntax below (exactly):

```
(Select Min(`Order Date`) & '-' & Max(`Order Date`)
from Orders
where `Customer ID` = Customer.`Customer ID`)
```

Place this field on the Group Header.
Note that is prints two dates - the first and last - for each Customer.
To extract these as date values for use in the report create the following two formulas:

Extract Min:

```
date( split({%SubQuery2},'-') [1] )
```

Extract Max:

```
date( split({%SubQuery2},'-') [2] )
```

SQL Commands

What is a Command

Crystal Reports (versions 9 - 12) allows you to write a SQL query, send it to the database, and use the results of that query as a table in your report. The statement is called a 'command' and it is saved with the report. A command is treated like a table, and can be linked to other tables or even to other commands. However, the most efficient method is (usually) to design one command that generates the entire result set required for the report.

Creating the SQL Command

To add a command you must either know how to write a SQL query or have access to a query that you can copy. One easy way to get a valid SQL query is to create or open a Crystal Report and copy the SQL shown by the menu options Database > Show SQL Query. When learning this feature it is helpful to have a SQL query sitting in notepad.

To add a command you start by opening the Database Expert, just as if you were going to add a table from a data source. But instead of selecting a table from the list of tables, you locate the item just above tables list - the one marked "Add a Command". This option appears inside every connection that is SQL based, so choose the one within the connection you intend to use for this report. A command can only be drawn from the data within one database connection.

When you double-click the "Add Command" item a window will open. This is where you type or paste your SQL statement. When you click OK at the bottom of this window your command will be validated by your database driver. If it is valid it will show up on the right hand side, as if it were a physical table in the database. From here on you treat it like any other table. The fields available within this command will be the list of fields in the SELECT of the SQL statement.

Making changes to the command

If you want to modify the command you need to reopen the Database Expert. You can right click on the command and select 'Edit Command' to open the command window. When you make changes and click "OK" your modified command will immediately be sent to the database to be validated and run. There is no prompt to ask you if you want to refresh.

Options not available when using Commands

Unlike the SQL in most reports, the SQL in a command is not affected by anything you do in the report design. The only way to change the SQL you enter in a command object is to go back into the Database Expert and edit the command SQL manually. These three features are disabled in a report that uses Commands:

Select Distinct
Perform Grouping on Server
All SQL Expression fields

Of course, most of these can be replicated within the SQL of the command.

Even changes to the Selection Formula, which would normally change the WHERE clause, have no effect on the SQL of a command. You can still use the record selection formula to reduce the number of records in the final report, but these added rules are only applied locally within Crystal, and are not incorporated into the SQL request sent to the database. The data set returned by the server will still be based on the criteria in the SQL Command query.

The down side of using commands

There are 3 things to keep in mind before you decide to use commands as the basis of a report:

First you have to know how to write or generate the appropriate SQL for your environment. If you make a mistake the error messages aren't always much help. I usually 'cheat' by constructing a Crystal Report that has the fields, joins and criteria that I want, and then copying the SQL from the "Show SQL" window to use as the starting point of my command.

Second, the SQL that works in one database will not always work in another. This forces you to learn the idiosyncrasies of your environment's Syntax. You will probably need to make changes if you need to use the same command with another database. See Appendix A for differences in basic tasks between the four SQL environments that I reviewed.

Third, there is no easy way to use a multiple value parameter in a command. All command parameters are single value. Ranges can be done by combining two single value parameters like Start Date and End Date. There are two workarounds for adding a multiple value parameter to a command that is shown in the examples.

Adding parameters to commands

There are special parameters just for commands that allow you to embed user selected values directly into the SQL query. You can add these prompts when the command is created, or any time you edit the command. On the right hand side of the command window you will see any existing parameters and buttons labeled, “Create”, “Modify” and “Remove”.

If you click the “Create” button Crystal will ask you to set four properties for this parameter. These properties are virtually the same as the properties in report parameters, but these parameters usually have to be single value parameters. They cannot be set to be ranges in any version of CR, and only CR 2008 (v12) has the “allow multiple values” option for command parameters. You can still create a range parameter, like a date range, by using two separate parameters (for StartDate and EndDate). There is no easy way to get a multi-value parameter directly into the SQL unless you are in CR 2008. See the two workarounds listed below for older versions.

Once you have saved a new parameter you must add it into the SQL for it to have any affect (or even for it to be saved). Place your cursor in the SQL at the point where the parameter is needed and double click the parameter in the list. The parameter will then appear in your command.

Now, each time you refresh the report, the parameter will prompt you for a value. The value you enter will be inserted into the SQL at the specified place. The most common place for command parameters is the WHERE clause, but this is not the only place where they can be used. You can also use parameters to replace table aliases, join types and fields in any part of the SQL (even fields in the SELECT if you add an alias - see below).

These parameters do a very simple substitution, inserting the value entered by the user into the SQL at the specified spot. The parameter doesn’t make any adjustments. So, for instance, if you put a string parameter into the WHERE clause you must remember to put the single quotes on either side of the parameter object, or you will get an error. These quotes are placed just outside the brackets of the parameter name. You could even add wildcard characters such as ‘%’ inside the quotes to make a parameter into an automatic ‘contains’ selection.

Enhancing Command Parameters

The parameters added in the command automatically appear in the list of parameters in the Field Explorer. Here you can rename the parameter, add a list of default values, set value limits, etc. However, if you go back the Database Expert and edit the command or its parameter in any way you will lose all of those changes. The report parameter will even be renamed back to the name used in the command. Any formula using the alternate name will now generate an error because the name has reverted. I recommend that you rename the command parameter and then make the corresponding changes to the command. If you remove a parameter from a command, that parameter will be automatically deleted from the list of command parameters as soon as you save the command. It will also be deleted from the list of report parameters.

Workarounds for creating a multi-value parameter in a command

As mentioned above there is no way to have a command parameter accept multiple values unless you are using CR 2008 (v12). However there are two workarounds that (in a limited way) allow the user to enter multiple values for use in the command.

The first can be done in any command based report. But it requires that the user be very careful. You create the command and add a single value parameter. Then in your WHERE clause you would use IN instead of an equal sign. And, instead of putting single quotes around the parameter name (in the WHERE clause) you put parentheses around it, so it looks like this:

```
WHERE quantity in ( {?QtyParam} )
```

Then, the user is instructed to enter their list of values into the parameter with commas between them. If the field in the WHERE clause is numeric, the entered values should look like this:

```
12, 14, 15, 16
```

However if the field in the WHERE clause is a character field, the user has to enter single quotes around each value like this:

```
'red' , 'blue' , 'green' , 'yellow'
```

Note that in both cases above the command parameter is set to be a ***string*** parameter, even if the field in the command is numeric. All command parameters are simple text substitutions made to the SQL, so the data type of the parameter does not have to match the data type of the corresponding field.

The weakness of this method is that it is very easy for the user to mess up, and it is not easy for you to prevent the mistakes. But this method can be used with any report.

The second method builds on the method above but allows the user to fill in a normal multiple-value parameter. It accomplishes this by making your report a subreport within another report. That means that this method can only be used in reports that do not already contain subreports. Here are the three steps:

1) Create a simple container report and add a multiple-value string parameter. Again, even if the values you want the user to enter are numbers the parameter can be a string, because it is going to end up in a command.

2) Use one of the following formulas to convert the multiple-value parameter of the main report into the string of values that you need to insert into the SQL statement.

For numbers use something like this:

```
Join ({?Orders} , ',')
```

For strings use something like this:

```
"'" & Join ({?States} , "','") & "'"
```

Note that on the second example the literal single quotes used by SQL need to appear in the final formula result. That is why they are nested within the double quotes of the Crystal formula.

3) Link the command parameter in the subreport (the one you created in the first method) to the formula in the main report. The formula takes the individual user selections and automatically formats them into the string that the user would have had to enter.

Thanks to Joel Seguin, Sr. Analyst at Health Net of California, for suggesting the subreport method and providing initial drafts of the formulas.

Storing the command in the repository

(Note - the repository is only available if you are using v9 or using Enterprise.)
The Enterprise level of Crystal Reports/Business Objects allows you to store certain types of objects into a central repository, which makes these objects easy to reuse. One type of object you can store is a command.

While you are in "Edit" mode you will see a check mark at the bottom of the window. If you check this box before you click OK, Crystal will add a copy of this SQL command to the repository so that it can be used to create other reports. If you have a repository configured and you have access to it, a window will open which will allow you to give this command a name. You can make final changes before it is saved.

The "location" box at the bottom of the window allows you to place this command into a specific folder. You can use the existing "Command" folder, a subfolder within this folder, or you can create a new folder. To select an existing folder you highlight that folder before saving the command. To create a new folder, you right-click on the folder marked "commands" and select the option 'new folder'. You click OK when you are ready to save the command into the repository.

Exercise 4 - Creating and Using Commands

4A) Simple Command
Open the Master Report.
Save it as Lesson 4A.
Change the select expert to say:
Customer.Region = CA
Use **Database > Show SQL Statement.**
Copy the SQL statement and paste it into Notepad.
Save and close the report.
Start a new report using the Standard Wizard.
At the "DATA" step, open the current connection to Xtreme.
Double click the option marked "Add a Command".
Paste the SQL statement into the box on the left and click OK.
Click "Next" to get to the "Fields" step.
Move all six fields over to the right hand side.
Click "Next" and on the Grouping step, add a group by Customer Name.
Click "Next" to get to the Summary step.
Remove the Customer ID; create a Count of Order ID and a Sum of Amount.
Click Finish to preview the report.
Note that the select criteria isn't needed because you have a WHERE clause in the SQL.

4B) Adding a Parameter

Open the Database Expert using **Database > Database Expert.**
Right-Click on the Command in the right window and select "Edit Command".
Click the Create button on the right to create a command parameter.
Fill in the four boxes as follows:

State
Enter a State Abbreviation
String
CA

Click OK.
Highlight the letters CA in the SQL Query window (don't highlight the quotes).
Double click the State parameter on the right to insert it into the SQL.
The parameter brackets should be between the single quotes.

Click OK and the prompt will appear.
Click OK again to return to the database expert.
Click OK again to return to the report.
Click the refresh button and select "Prompt for new parameters".
Enter PA and click OK.
Notice that the report only includes PA records.
Save this report under the name Lesson 4B.

4C) Passing a multiple value parameter to a command

Open the Master Report.
Save it under the name "Lesson 4C".
Unsuppress the Report Header and suppress all other sections.
Create a new parameter called "States".
Check "Allow Multiple Values".
Make sure you select 'discrete' values and not 'range' values.
Click Default Values and type CA into the "Select or Enter Value" box.
Click the arrow to move it to the Default Values list.
Add FL, PA and OH to the list as well (one at a time).
Click OK to exit the parameter.

Use the menu commands Insert > Subreport.
Select "Choose an existing report" and click Browse.
Locate Lesson 4B from the previous lesson and open it.
Click OK and place the subreport into the report header.
Widen the subreport to be the full page width.

Right-click on the subreport object and select “Edit Subreport”.
Go into the Database Expert, right click on the command and select “Edit Command”.
Change the WHERE clause to be:

`Customer`.`Region` in ({?State})

(The equal sign changes to IN and the quotes around the brackets change to parentheses.)
Highlight the parameter ‘State’ and click Modify.
Change the default value to ‘CA’ (putting single quotes around the value).
Click OK, accept the new default value and click OK twice to get back to the subreport.

Switch back to the design screen of the main report.
Create a formula field called “States Link” that looks like this:

```
"'" & Join ({?States} , "','") & "'"
```

Note the double quotes used around all of the single quotes.

Right click on the subreport object and select “Change Subreport Links”.
Find the formula “States Link” and move it to the box in the upper right.
In the lower left box, select the “State” parameter in the subreport.

(Don’t select the default “?PM-” parameter that appears by default.

Refresh and select “Prompt for new parameters”.
Add all four states to the list.
Note that all four of those states appear in the command subreport.
Save as lesson 4C.

4D) Storing the Command in the Repository (v9/Enterprise with a Repository only)
Open Lesson 4C.
Open the database expert using **Database > Database Expert.**
Right-Click the command in the right hand window and select “Edit Command”.
Place a check mark in the lower left corner next to “Add to Repository”.
Click OK and give the command the name “Master”.
In the Location box below, open the repository folder.
Right click on the “Command” folder and select “New Folder”.
Enter the name “TEST” for the new folder.
Highlight that folder and click OK to put the Master command into this folder.

Common problems that can be solved by Command Objects

Creating UNION queries

The most common reason I use commands for my clients is when I need to do a UNION query. It could be that they need to append records from one table to records from another. Or it could be that you need to append a table to itself. UNION queries can be used in both of these situations.

Here is a common example. Say you have employees in one table and customers in another. To show all of these people in one report, alphabetically, you need to append the employees onto the customers and then sort the resulting table. The first step is to create two separate queries, one from each table. Both queries should have corresponding fields, with the same data types, and be in the same order. Here is the SQL for both simple queries:

```
SELECT CustID, CustNAME
FROM Customers

SELECT EmpID, EmpName
From Employees
```

To UNION these you simply put the word UNION between them and make sure that the field names you want are in the first query. If you don't want the field names from either query you can give those fields an alias. So my UNION would look like this:

```
SELECT CustID as ID, CustNAME as Name
FROM Customers

UNION

SELECT EmpID as ID, EmpName as Name
From Employees
```

The resulting "table" generated by this UNION has two columns, ID and Name.

Using the word UNION will eliminate duplicate rows, whether they come from within a single table or from the combined tables. If you need to preserve duplicates you should use UNION ALL.

If you want to identify the source table for each record you can add a column of literal values to each query with an alias field name. The literals below will appear in a 3rd column called "source". The alias names are typically optional in the second query, and are set in the first query.

```
SELECT CustID as ID, CustNAME as Name, 'CUS' as source
FROM Customers
UNION
SELECT EmpID, EmpName , 'EMP'
From Employees
```

Filtering on an Outer Join

As we discussed above, an Outer Join means that one table is optional. You will get records from the inner table even if there are no matches in the outer table. One of the limitations of an Outer Join is that there can't be any filtering done on the outer (or optional) table. If you put any criteria on the outer table the Outer Join will 'break' and behave like an Inner Join. Records in the inner table that have no match in the outer table would drop off the report.

This happens because, in standard SQL, the tables are joined before the filter is applied. The filter is therefore applied to the records of the combined results table. But a command can create a table of filtered results before the join occurs. You could then do an Outer Join to those results. Because the filter is applied to the command before the join occurs, the Outer Join isn't affected by the filter.

So say you want a list of all employees, showing the list of the orders that they have processed year to date. You want the report to include all employees - even those that don't have any sales this year. So you join from the Employee table to the Orders table. You make that a Left Outer Join so that the orders table is optional. This works fine until you add the filter for year to date orders. A filter on the Orders table means that this table is no longer optional. The Outer Join breaks. The report will now include only employees that have orders this year.

Why adding an IsNull() to the filter doesn't always work:

I have had many users suggest a seemingly simple solution to this problem. They add an IsNull() to the filter so it says that the filter field can either be NULL or meet the criteria. This is deceptive because it solves only a small piece of the problem. Take my example above where we have employees and their sales for several years. The IsNull() solution would be a filter that says:

IsNull ({Sales.OrderDate}) or {Sales.OrderDate} in YearToDate

Now think of the employees in the following four categories:

A) Having sales in both the current year and the prior year
B) Having sales in the current year, but not in the prior year
C) Having sales in the prior year, but not in the current year
D) Having no sales in any year

Categories A and B have sales this year so they are included. Category D will have no sales records so the Outer Join will generate one NULL sales record, and those employees will be included because they qualify under the IsNull() rule. However, category C still doesn't qualify. Because they have old sales records there is no NULL record for them. So they don't qualify under the IsNull(). And their sales are all before the period so they don't qualify under the second part of the filter. So employees in Category C will still be excluded.

Two possible Solutions
A) You could remove the Orders (outer) table from the report and replace it with a command. The command would query the Orders table and use a WHERE clause to limit the records to the current year. If you join the Employee table to this command with a Left Outer Join, the filter will take place before the join. There will be no affect on the join. However, when you link a command to a table you force some of the processing to occur locally in Crystal. This can significantly affect performance.

B) Instead you could do the entire query in one command. You do this by moving the filter for the outer table from the WHERE clause to the FROM clause, like this:

```
SELECT
`Customer`.`Customer Name`, `Orders`.`Order ID`,
`Orders`.`Order Amount`, `Orders`.`Order Date`
FROM `Customer` LEFT OUTER JOIN `Orders`
ON (
`Customer`.`Customer ID`=`Orders`.`Customer ID`
and  `Orders`.`Order Date`>={ts '1997-01-01 00:00:00'}
and `Orders`.`Order Date`<{ts '1997-05-10 00:00:00'}
)
WHERE `Customer`.`Country` = 'USA'
```

In the example above I have left one filter in the WHERE clause because it is from the INNER (Customer) table. But I moved the date range criteria to the FROM. Now the report will return all customers in the USA regardless of what sales they have. If I had left the date range criteria in the WHERE I would have broken the Outer Join and would have only returned the customers who had sales in that date range. (Note that putting the parentheses around the ON part of the join is only required in MS Access. It is optional in the other 3 SQL environments that I tested.)

Exceptions that I have found
In the following situations a Left Outer Join may not break when filtered:

1) If you are using the SQL Server native client (which is only supported in older versions of Crystal). SQL Server's native client didn't always follow standard SQL and would in some cases apply the filter to the table before creating the join.
2) If you link tables from 2 different databases then Crystal will have to run 2 separate queries and merge them together locally. This means the filter will happen before the join. This is also forcing much of the work to happen locally.

The last technique (filtering in the FROM clause) was suggested by Linda Bass. Linda is currently the top Crystal Reports expert on Tek-Tips.com, where she goes by the handle of Lbass.

Using a summary query (Group By) to prevent table inflation

Most links between tables involve a one-to-many relationship. If you try to link two different "many" tables to the same "one" table you will get a dataset that is very difficult to work with. This is especially true if you are trying to do totals from both of the "many" tables. One solution is to take one of the two "many" tables and create a command that groups by the linking field, and calculates the needed totals in the command. Since the command will only have one record for every group of the main report, adding a join to the command won't additionally inflate the report data.

Take a list of invoices that have line item charges in one table and credits (like payments and adjustments) in another. Because there could be multiple credit transactions for the same invoice it would be difficult to just link the invoice records to both their line items and their payments at the same time. An invoice with four line items and two credit transactions would generate eight records and both the line items and the credits would be duplicated. It would be very difficult to get totals from either table. But if all you need is the total of the credits you could write a query that returns the total amount of credits for each invoice - one record per invoice. You could then link that summary results table to the invoice table, making it possible to use those numbers without inflating the line item totals in the original report.

This requires a GROUP BY clause in our query. The syntax for a GROUP BY requires that every field in the SELECT be either one of the group fields or a summary field. Here is an example form the Xtreme database:

```
SELECT `Orders`.`Customer ID`,
    COUNT(`Orders`.`Order ID`),
    SUM(`Orders`.`Order Amount`)
FROM   `Orders` `Orders`
WHERE  `Orders`.`Order ID` < 2000
GROUP BY `Orders`.`Customer ID`
ORDER BY `Orders`.`Customer ID`
```

The only field in the SELECT that isn't a summary is the group field Customer ID. This command would only return one summary record for each customer. That means that it could be added to any report and, if linked by Customer ID, would not cause any table inflation.

Note that summaries like these could also be calculated in a SQL Expression with a correlated subquery. With a concatenate, you could even bring back both summaries in one expression. It is hard to say which would be faster. That would require testing the two methods side by side with a real example. But some things to consider would be:

- SQL Expressions are performed completely on the server while a linked command is run on the server but linked to the rest of the report locally.
- SQL Expressions are subqueries which have to be performed recursively while a command is run as a single query.
- SQL Expressions can't have parameters while Commands can.

Allowing users to change tables, join types and fields by Parameter

Commands allow for parameters just like reports. But command parameter can do things that you can't so with report parameters, because they can be used to replace virtually any text in the SQL Query. This means that they can:

1) Replace table aliases - giving the user a choice of which table to query.
2) Replace a Join Type - allowing the user to select Outer versus Inner Joins
3) Replace Summary Functions - Choosing Min or Max in a calculation
4) Add the word Distinct at runtime - eliminates duplicate rows
5) Replace field names - allowing field choice directly in the SQL

There is one note of caution if you decide to replace the field name in SQL SELECT statement. The names of the fields returned to the report can't change or the report will have to verify and map fields. To prevent this you should create an alias for the field, right after the parameter. Here is a simple example where the parameter allows the user to choose from several date fields. Whatever they choose is given the alias "Date" so that the report always sees the same field name.

```
SELECT {?FieldParam} as Date, ID, Amount
FROM Orders
WHERE ….
```

Enhancing Dynamic Parameters (Versions 11 and 12 only)

One of the primary enhancements of version 11 was ability to create a dynamic parameter, meaning a parameter that let the user choose their value from a dynamically generated list of values. The list can be generated from either a column in a table or a column in a command. The advantage of using a command is that you can add a few features to your pick list that you can't add by going directly to a table. With a command you can:

1) Filter specific values by adding a WHERE clause to the command
2) Use a UNION to add custom values to the list, like the word "ALL".
3) Use expressions to build a more sophisticated description column.

You can write the command using any SQL you want, and then you add it to the report just like any other table. Note fields from the command that are intended to feed a parameter don't have to be used anywhere else on the report. The command doesn't even have to be linked to the other tables (although Crystal complains when you leave tables unlinked). Even if left unlinked it will be available for generating your parameter values. And, if you want to pull both values and descriptions from the command then make sure your command returns both the value and the corresponding description field(s).

Exercise 5 - Solving Common Problems with Commands

5A) Creating a Union

Open Lesson 5A1.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Show the SQL and copy it.
Open a text editor (WordPad, Notepad, Word) and paste the SQL into an empty file.
Type the word UNION below the SQL query.

Open Lesson 5A2.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Show the SQL and copy it.
Paste this SQL below the word UNION in your text editor.

After each field name in the SELECT add an alias so that it looks like the statement below.
(Putting each field on its own line isn't essential but makes it easier to read.)
(Also note that the aliases are optional in the second query.)
Add a literal value line to each query to distinguish Employees (EMP) and Customers (CUS):

```
SELECT
`Employee`.`Employee ID` as ID,
`Employee`.`Last Name` as FName,
`Employee`.`First Name` as LName,
'EMP' as type
 FROM   `Employee` `Employee`

UNION

SELECT `Customer`.`Customer ID` as ID,
`Customer`.`Contact First Name` as FName,
`Customer`.`Contact Last Name` as LName,
'CUS' as type
FROM   `Customer` `Customer`
```

Don't confuse the back ticks around field/table names with the single quotes around strings.

Now start a new report using the Xtreme Database, but don't add any tables.
Instead use the "Add Command" option, just above the "tables" category in the Database Explorer.
Copy the full query from your text editor and paste it into command.
Click OK and add all four fields from the command to the report.

When you run the report you should have a list that includes all employees and all customers.

5B) Filter an OJ
Open the report Lesson 5B1.
Refresh the report, connecting it to your Local Xtreme Sample connection.

Open the report Lesson 5B2.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Look at the dates on the first page and note one of the years returned for a later step.
Go to the database menu and select Show SQL Query.
Copy the query and close 5B2.

Go back to 5B1 and open the Database Expert.
Add a Command to the report and paste in the SQL you copied.
Add the following line to the bottom of the Query (use the year number noted above).

```
WHERE Year (`Order Date`) = 2004
```

Click OK in the Command window.
Then Click on the LINKS tab.
Right-click on the line between tables, and select Link Options.
Change the link type from Inner to Left Outer.
Click in the background and see if the arrow points from Employee to Command.
If not, right-click on the arrow and select “Reverse Link”.
Click the “Arrange Tables” button to confirm that the Employee table is on the left.
Click OK four times to close the Database Explorer and Refresh the report.
Open the Field Explorer and add the Order Date field from the command to the report.
Note the record count immediately jumps as the original table is inflated.

Note that this report includes only sales for one year.
Jump to the last page.
Note that this report also includes employees with no sales.
You still have a Left Outer Join but the outer table has had a filter applied.
Save the report as Lesson 5B3.

5C) Using a Summary GROUP BY table to reduce inflation
Open Lesson 5C1 which shows orders for each customer.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Show the SQL and notice there is no GROUP BY clause.
Hide the details and the Group Header.
Go to File > Report Options and check “Perform Grouping on Server”.
Show the SQL and notice there is now a GROUP BY clause.
Copy the SQL.

Open Lesson 5C2, which shows credit transactions for each customer.
Go into the Database Expert and add a new Command.
Paste in the SQL from Lesson 5C1.
Click OK to close the Command window.
Click on the Links tab.
Link from the Customer table to the Command using Customer ID.
Go into the Link Options for this Join and change it from Inner to Left Outer.
(If Crystal creates the link, make sure the arrow points from Customer to the Command.).
Click OK 3 times to refresh the report.
Open the Field Explorer.
Put the fields EXPR 1001 and EXPR 1002 onto the Group Footer.
Note that each Customer now has a summary of both the credits and the orders.
Save as 5C3.

5D) Change other parts of SQL Query
Open up Lesson 5D1.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Save the report under the name Lesson 5D2.

Open the Database Expert, right-click on the Command and select Edit.
Click the “Create” button on the right to create a new parameter.
Name the parameter “Join Type”.
Make the Prompting Text “Select Inner or Left Outer”.
Set the default value to be INNER.
Click OK to close the parameter window.
Highlight the word INNER in the FROM clause of the command.
Double-click the parameter name.
The parameter name should replace the word INNER in the query.
Click OK and you should be prompted by the parameter.
Click OK to accept the default value of the parameter.

Open the Field Explorer and find the new parameter called Join Type.
Right-click on this parameter and select Edit.
Click the “Default Values” button.
Enter LEFT OUTER into the “Select or Enter Value” box and move that over to the right.
You should now have two default values to pick from.
Click OK twice to close the parameter design window.

Refresh the report and select “Prompt for new parameter values”.
If you pick LEFT OUTER the report has employees listed with no sales.
If you pick INNER you see only employees that have sales.

5E) Enhancing Dynamic Parameters (Versions 11 and 12 only)
Open up Lesson 5E1.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Save the report under the name Lesson 5E2.

Add a new dynamic parameter to select the Sales Rep:
Open the Field Explorer and right-click on the Parameters category.
Select "New" and name the parameter "Employee from Table".
Change the data type to Number and select Dynamic.
Click in the box that says "Click here to add item" and select Employee ID.
Click one box to the right to select Employee Last Name as the Description Field.
(Note that you can't select both first and last name).
Click in the third box to create the parameter.
Click OK to save this parameter.

Switch to Design mode.
Open the Select Expert and add a second rule that says:

```
Employee ID / is equal to / {?Employee from Table}
```

(Select the parameter from the drop down list).
Click OK and refresh the report.
Ask it to prompt for new parameter values.
Note that the list includes ALL employees, not just Sales Reps.
Select #9, Dodsworth.
Click OK and the report will return only records for Dodsworth.
Save this (again) as Lesson 5E2.

Open up Lesson 5E3.
Refresh the report, connecting it to your Local Xtreme Sample connection.
Show the SQL and copy it.
Close the report.

Go back to Lesson 5E2 and open the Database Expert.
Add a new Command from the current connection.
Past in the SQL that you copied from above.
Replace the comma between Last Name and First Name in the SELECT clause with this:

```
& ', ' &
```

After `First Name` add:

```
as name
```

Click OK 3-4 times to close the Command window.
Click "Cancel" when it asks to refresh.
.

Add a second dynamic parameter just like the first.
Name this parameter “Employee from Command”.
Change the data type to Number and select Dynamic.
Click in the box that says “Click here to add item” and select Employee ID from the Command.
Click one box to the right to select Name (from the command) as the Description Field.
Click in the third box to create the parameter.
Click OK to save this parameter.

Switch to Design mode.
Open the Select Expert and change the second rule to say:

```
Employee ID / is equal to / {?Employee from Command}
```

(Select the parameter from the drop down list).
Click OK and refresh the report.
Ask it to prompt for new parameter values.
Note that the list includes only Sales Reps.
Also note that it includes the “Name” expression you created in the command.

Modifying the Auto-Generated SQL (v8.5 and prior)

If you are still using an older version (anything up through v8.5) you have the option to make limited changes to the SQL by going to Database > Show SQL Query. You are allowed to change any section of the query except for the SELECT clause. In general these changes will take precedence over what you do in the report features. However, some report features will still interact with the SQL. Here are some examples of the interaction between manual changes to the SQL and changes to the report:

Selection

If you add a rule to the WHERE clause using a field that is not currently used in the Select Expert, the Select Expert is not affected and new rule is added to the criteria. However if you change or delete a rule in the WHERE clause, and that rule IS used in the Select Expert, then the Select Expert will be completely cleared and the WHERE clause in the SQL will be the only criteria. At this point new rules added to the Select Expert will generate a warning that the SQL has been modified. Rules added in the Select Expert will not show up in the SQL but will be applied locally.

Note, if a report has customized SQL and you deleting all the rules in the Select Expert may generate an error message because the word WHERE may appear with no rules. Deleting the word WHERE will clear this error.

Joins

Changing a join in the SQL will be passed to the database but will not be reflected in the Visual Linking Expert options. Once manual changes are made to the joins, changes made in the Visual Linking Expert are stored, but are no longer passed to the SQL.

Sorting and Grouping

Changing the ORDER BY in the SQL will override the fields in Sort Records. Subsequent changes in the Sort Records will affect the sort temporarily until the next refresh. At refresh the record and group order will revert to the order specified in the SQL.

UNION Queries

As long as you don't change the initial SELECT you can add the word UNION and a second query in the SQL window. The second query will have to have fields of the same data types and in the same order, as those found in the original SELECT. Since you can't change the initial select your report will show the column names of the report's original SELECT. You can UNION as many queries as you want this way. And since you can't change the first SELECT you actually have more flexibility writing the second or third query in the UNION than you do the first.

For this reason you might want to create a report where the first SQL statement does nothing but define the columns. Any list of fields with the appropriate data types, in the correct order will do. You can even make the WHERE clause of this query a false value like "1 = 0". This ensures that this query won't add any records to the results. You then have free reign to write one or several queries, using any valid SQL for your database, as long as it returns the same number of columns with the same data types in the same order.

The other option for gaining full control of the SQL is to use the SQL Designer discussed below, or upgrade to a newer version of Crystal Reports that supports the use of Command objects.

Exercise 6 - Modifying SQL Queries in v8.5

Open the Master Report.
Go to Report > Select Expert to see the existing criteria.
Click OK.

Go to Database > Show SQL Query to see the existing SQL.
Add a line to the WHERE clause that says:

```
and Region = 'CA'
```

Click OK to see that only CA records are returned.
Note the record count at this point.
Open the Select Expert and note that the original rule is still there.
Change the value 1800 to the value 2000.
Click OK and refresh the report.
Note the record count drops.

Show the current SQL.
Change the value 2000 in the WHERE clause to be 1000.
Click OK to see the increased records.
Open the Select Expert to see that there are no rules listed.

Add a rule in the Select Expert that says:

```
Order Amount  / is greater than / 1000
```

Click OK and note the message that appears.
Refresh the data and note that only records over 1000 appear.
Show the SQL and note that this last rule is not in the SQL.
(It is applied to the data after it is returned from the database.)

Using Crystal SQL Designer (v8.5 and prior)

If you want to use a SQL statement that is beyond what Crystal Reports can generate, you can use the SQL Designer to create a Crystal "Query" file. You can create or paste any valid SQL statement into the Query, and then use the Query as the source of one or more reports. The SQL designer also allows you to save a Query with a "snapshot" of the data it returns. This saved data can then be used as the source for reports. With saved data, the reports no longer connect to the server, but run using only the "snapshot" of data saved in the Query.

The Crystal SQL Designer is a separate utility found in the program menu. It can also be run using CQW.exe from Windows Explorer. It may not be installed by default so you may need to reinstall Crystal to add the SQL Designer to your PC. Once you create a Query and save it, it has the extension QRY. This QRY file always contains the SQL statement. Once it has been run you have the option of saving it with the data that it returned from the database. In either case the QRY file becomes the data source for your report.

Running a report using a QRY Without saved data

If you create a report using a QRY *without* saved data, the report will run the SQL statement and get data from the database. It will use this data in the report. If later you hit the refresh button in the report, the SQL statement will run again, getting live records again from the database.

Running a report using a QRY With saved data

If you create a report using a QRY *with* saved data, the report will not need to run the SQL statement. The report will, instead, read the data stored in the QRY file. Your report will show the data that was extracted when the query was last saved. Hitting refresh in the report will, in most cases, not change the report. This is because it will only refresh against the data stored in the QRY. It will not cause the QRY to query the database. Because this QRY has saved data, you can only update it from the SQL Designer. You would have to rerun it within the SQL Designer and then resave it with its data. You can then rerun the report that uses this QRY and see the newer records.

A QRY is useful when you have multiple reports that require the same data. You can create a QRY that collects the appropriate records, and design several reports to use that same QRY. This will lessen the burden on your data server, since you only need to extract the data once. You can also use a QRY to create a portable data set that you can:

1) Take on the road with you.
2) Share with other people who need the same records.
3) Archive as a record of how the data stood at a particular time.

Miscellaneous Notes for Query files

- Queries can't be joined to any other tables in a report, except through a linked subreport.
- Queries are always fully "inflated" by their one-to-many joins, regardless of the fields used.
- DateTime fields will become strings, but you can use DTSToDateTime() to convert them back.

Creating a QRY file

1) **In Version 8** go to the start menu option labeled “Seagate Crystal Reports Tools”. Select the option called “Crystal SQL Designer”.
   In Version 7 go to the start menu option labeled “Seagate Crystal Reports 7”.
   Select the menu option labeled “32-bit Crystal SQL Designer”.
2) Once you have loaded the SQL Designer select the menu options **File > New**.
3) Click on the button marked **Use SQL Expert.** (If you are pasting in a copy of an existing statement you can skip to the last tab.)
4) Select Tables, Links and Fields as you would in the report expert. On the fields tab, you can’t add formula fields, but you can add SQL expressions.
5) The sort tab allows you to sort the QRY records. Sorting or Grouping in the final report will override this sort. The only exception is if you set your group to be “in original order”, which creates headers and footers, but doesn’t change the order of the records. They stay in the order that they Query puts them in.
6) The select tab allows you to specify which records should be included in the QRY. This becomes the “Where” clause. Add rules just like you would within Crystal.
7) The SQL tab shows the initial SQL statement based on the previous tabs. This statement can be modified as long as the resulting SQL is valid for your database.
8) Click “Finish” at the bottom. Crystal will allow you to either save the QRY or run it. Before you can save data with the Query, you must run the Query.
9) To save data with the Query use the menu options **File > Save Data with Query.** Each time you select this menu option you toggle the check mark on and off. When you use **File > Save** it will check this setting to see whether to include the data. This setting is checked “on” by default so you may not need to change it. If you don’t want saved data, you must remove the checkmark.
10) Save the Query using **File > Save**, noting the location of the QRY file. You will need to find this file when you create your report. If you saved the QRY with data, it is portable and no longer needs to access the database.

Creating a Report from a QRY file

1) Start the report expert as you normally would.
2) On the data tab, select the option “Crystal SQL Query”.
3) Locate the QRY file to use, and select it.
4) It will show only one table called “Query”.
5) On the “Fields” tab you will see all of the available fields in the query.

From this point forward the report expert will behave the same as any other report.

Exercise 7 - Using the Crystal SQL Designer

Creating a QRY file:
Start the SQL Designer from the Crystal Tools menu.
Select the menu options **File > New** and click the top button "SQL Expert".
USA the "SQL/ODBC" button to connect to "Xtreme Sample Database".
Select two tables, Customers and Orders, and link them.
From Customers take the fields Customer Name and Region.
From Orders take the fields Order ID, Order Amount and Order Date.
On the Sort tab, sort the fields by Region.
On the select tab, select Country is equal to USA.
The last tab will show you your SQL statement.
(If you know SQL you could edit this statement.)
Click "Finish" and allow the SQL designer to "Process the query".
You will see the result set that is returned based on your SQL statement.
Click the "File" Menu and look at the options in the list.
Confirm there is a check next to "Save Data with Query", if not click that option.
Select the menu options **File > Save** and save the QRY as WithData.qry.
Select the menu options **File > Save Data with Query** to remove the checkmark.
Select the menu options **File > Save As** and save the QRY as NoData.qry.

Close the SQL Designer and open windows Explorer.
Locate the two QRY files and note the difference in size (one includes data).

Create a report off of these QRY files:
Start Crystal Reports and on the Data tab, select the "Crystal SQL Query" button.
Locate WithData.QRY and select it.
On the fields tab select just the field Customer Name and preview the report.
Note that you still get an inflated result set with many records per customer.

Use the menu commands **Report > Report Expert.**
On the Fields tab, select all fields.
On the Group tab, group on "Region" and set the order to "In original order".
On the Total tab, change "Order ID" to a count.
Preview the report.
The sort of the query is now controlling the group order.
(In this case, it would still look the same if the report was controlling the group order.)
Click on the date and select **Format > Format Field**, note that it is a string.
Add a formula called "RealDate" using the formula:

```
DTSToDateTime ( {Query.Order Date} )
```

Place this field on the report and use **Format > Format Field.**
Note that you now have date formatting options for this field.

Appendix A - SQL syntax variations

Field or table names with spaces

MS Access uses backticks	`` `my table`.`my field` ``
MySQL uses backticks	`` `my table`.`my field` ``
Oracle uses double quotes	`"my table"."my field"`
DB2 uses double quotes	`"my table"."my field"`
SQL Server uses square brackets	`[my table].[my field]`

Note that Crystal will punctuate all table and field names with these markers, even those that don't have spaces. But they are only needed when there is a space in the middle of the name.

Punctuation for Literal Strings

All use single quotes around literal strings.

Punctuation for Literal Dates

MS Access:	#12/31/2007#
My SQL:	'2006-04-02 01:09:00' (implicit conversion)
Oracle:	Cast ('01-JAN-2004' as date)
DB2:	Cast ('01-JAN-2004' as date)
DB2:	Date('2007-12-31')
SQL Server:	Cast ('01-dec-2007' as datetime)

Syntax to Concatenate strings

Oracle:	string1 || string2 (any number of items)
Oracle:	CONCAT(str1,str2) (only two items)
DB2:	string1 || string2 (any number of items)
DB2:	CONCAT(str1,str2) (only two items)
SQL Server:	string1 + string2
MS Access:	string1 + string2
My SQL:	{fn CONCAT(status, 'abc' , 'dedf')} (any number of items)

Appendix B - Other SQL Resources

Syntax Comparison for nine different flavors of SQL
http://sqlzoo.net/howto/source/z.dir/

Oracle vs SQL Syntax :
http://www.bristle.com/Tips/SQL.htm

Oracle Syntax including a list of functions:
http://ugweb.cs.ualberta.ca/~c391/manual/chapt6.html

Appendix C - Reporting on Views and Stored Procedures

What they are

A view is a combination of tables that are linked together and includes selected fields. A stored procedure can go further by combining several different queries and other data manipulations. Both of these can be used by Crystal to provide data for a report.

When they are eligible

Typically all views and most stored procedures will be available to use in Crystal reports. This is because all views and most stored procedure end by generating a table of values. If you add the object to the report (like adding a table) Crystal will launch that object to generate the results table. The results table will be treated by Crystal just like any other table.

Making them visible

If you don't see your views or stored procedures in your database expert it may be because they weren't enabled in your Crystal options. Go to File - Options and click on the database tab. Then look at the section marked "Database Explorer". Make sure the checkmarks for "views" and "stored procedures" are both checked. If you change the settings you should close and reopen Crystal Reports to see the change. If they still don't appear, you should check with your Database Administrator to confirm that your user name has the correct permissions set to see the views and stored procedures.

Parameters in a stored procedure

A stored procedure may have parameters built into it. If you use a stored procedure with a parameter in your report, the parameter will automatically appear in your list of parameters. These parameters can't be deleted from the report and most of the properties are locked. This is one way to recognize that a parameter is coming from the stored procedure. Otherwise they can be used like other report parameters.